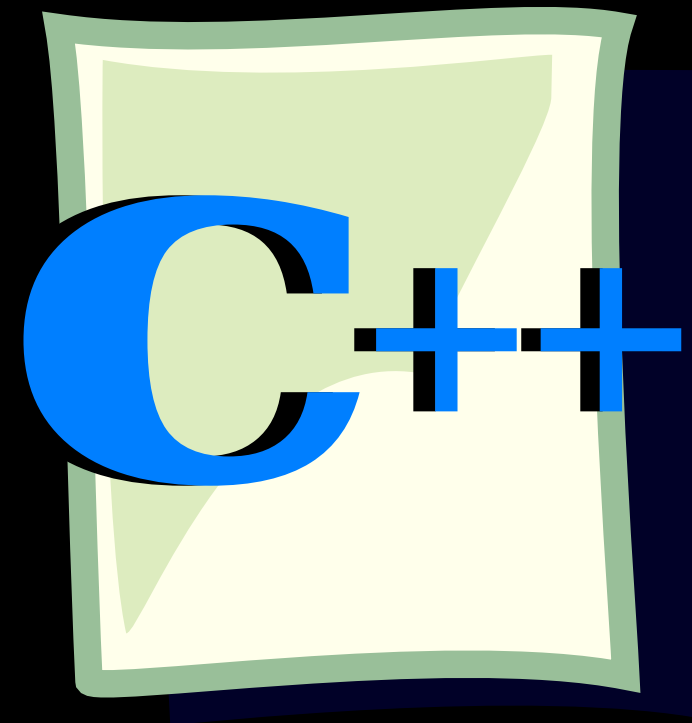


# *Variables, expresiones y operadores*



# Variables

# Variables

---

*Datos que se mantienen en memoria*

Variable: dato que se accede por medio de un nombre

Dato literal: un valor concreto

Variable: puede cambiar de valor (*variar*)

```
edad = 19; // variable edad y literal 19
```

Las variables deben ser declaradas

¿Qué tipo de dato queremos mantener?

- ✓ Valor numérico sin decimales (entero): tipo `int`
- ✓ Valor numérico con decimales (real): tipo `double`

Declaración: *tipo nombre;*

# Variables

---

*Declaración de variables*

*tipo nombre;*

`int cantidad;`

`double precio;`

Se reserva espacio suficiente

cantidad

precio

Memoria

?

?

...

LAS VARIABLES NO SE INICIALIZAN

No se deben usar hasta que se les haya dado algún valor

*¿Dónde colocamos las declaraciones?*

Siempre, antes del primer uso

Habitualmente al principio de la función

# Variables

## *Declaración de variables*

```
#include <iostream>
using namespace std;

int main()
{
    int cantidad;
    double precio, total;

    return 0;
}
```

cantidad

precio

total

Memoria

?

?

?

...

Podemos declarar varias de un mismo tipo  
separando los nombres con comas

# Variables

---

## *Capacidad de las variables*

int

-2.147.483.648 ... 2.147.483.647

-2147483648 .. 2147483647

double

$2,23 \times 10^{-308}$  ...  $1,79 \times 10^{+308}$  y sus negativos

[+|-] 2.23e-308 .. 1.79e+308 ← Notación científica

Problemas de precisión

# Variables

---

*Asignación de valores a las variables (operador =)*

`variable = expresión;` ← Instrucción: termina en ;

`cantidad = 12; // int`

`precio = 39.95; // double`

`total = cantidad * precio; // Asigna 479.4`

`cantidad ← 12`

Concordancia de tipos:

~~`cantidad = 12.5;`~~

*!!!A la izquierda del = debe ir siempre una variable!!!*

# Expresiones



# Expresiones

---

## *Expresiones*

Secuencias de operandos y operadores

*operando operador operando operador operando ...*

total = cantidad \* precio \* 1.18;

|  
Expresión

A igual prioridad se evalúan de izquierda a derecha

Paréntesis para forzar ciertas operaciones

total = cantidad1 + cantidad2 \* precio;

total = (cantidad1 + cantidad2) \* precio;

≠

Unos operadores se evalúan antes que otros

✓

# Expresiones

---

## *Precedencia de los operadores*

```
cantidad1 = 10;
cantidad2 = 2;
precio = 40.0;
```

\* y / se evalúan antes que + y -

```
total = cantidad1 + cantidad2 * precio;
```

\* antes que +  $\rightarrow 10 + 2 * 40,0 \rightarrow 10 + 80,0 \rightarrow 90,0$

```
total = (cantidad1 + cantidad2) * precio;
```

+ antes que \*  $\rightarrow (10 + 2) * 40,0 \rightarrow 12 * 40,0 \rightarrow 480,0$

# Variables y expresiones

variables.cpp

## *Ejemplo de uso de variables y expresiones*

```
#include <iostream>
using namespace std;

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
    precio = 39.95;
    total = cantidad * precio;
    cout << cantidad << " x " << precio << " = "
         << total << endl;

    return 0;
}
```

# Variables y expresiones

---

## *Ejemplo de uso de variables*

```
#include <iostream>
using namespace std;
```

```
int main()
{
    int cantidad;
    double precio, total;
```

cantidad

precio

total

Memoria

?

?

?

...

# Variables y expresiones

## *Ejemplo de uso de variables*

```
#include <iostream>
using namespace std;

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
```

	Memoria
cantidad	12
precio	?
total	?
	...

# Variables y expresiones

## *Ejemplo de uso de variables*

```
#include <iostream>
using namespace std;

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
    precio = 39.95;
```

	Memoria
cantidad	12
precio	39.95
total	?
	...

# Variables y expresiones

## *Ejemplo de uso de variables*

```
#include <iostream>
using namespace std;

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
    precio = 39.95;
    total = cantidad * precio;
```

	Memoria
cantidad	12
precio	39.95
total	479.4
	...

# Variables y expresiones

## *Ejemplo de uso de variables*

```
#include <iostream>
using namespace std;

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
    precio = 39.95;
    total = cantidad * precio;
    cout << cantidad << " x " << precio << " = "
         << total << endl;
```

	Memoria
cantidad	12
precio	39.95
total	479.4
	...

```
D:\FP\Tema2>variables
12 x 39.95 = 479.4
```



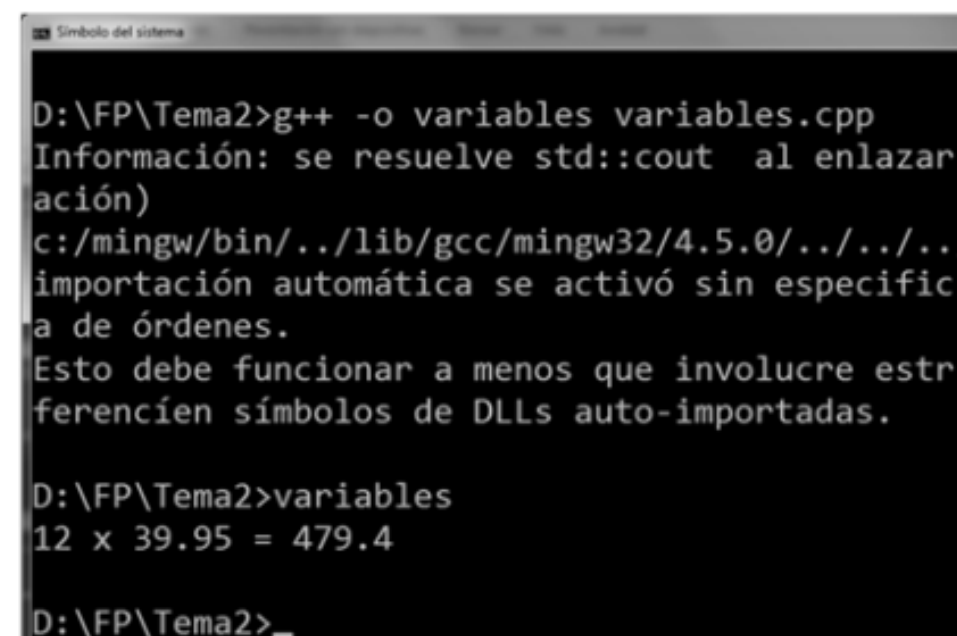
# Variables y expresiones

## *Ejemplo de uso de variables*

```
#include <iostream>
using namespace std;

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
    precio = 39.95;
    total = cantidad * precio;
    cout << cantidad << " x " << precio << " = "
         << total << endl;

    return 0;
}
```

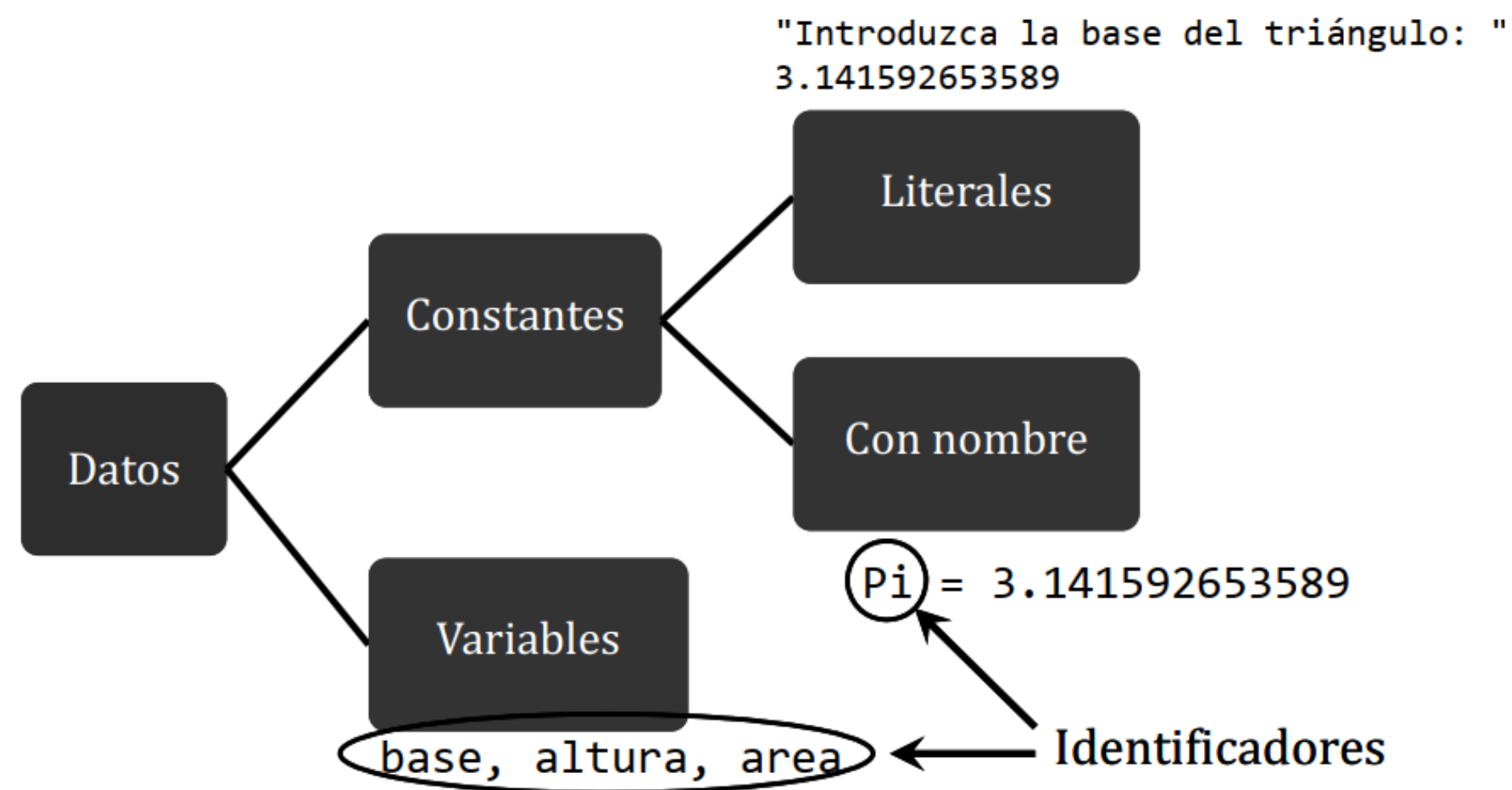


```
Símbolo del sistema
D:\FP\Tema2>g++ -o variables variables.cpp
Información: se resuelve std::cout al enlazar
ción)
c:/mingw/bin/./lib/gcc/mingw32/4.5.0/./../..
importación automática se activó sin especific
a de órdenes.
Esto debe funcionar a menos que involucre estr
ferencién símbolos de DLLs auto-importadas.
D:\FP\Tema2>variables
12 x 39.95 = 479.4
D:\FP\Tema2>
```

# **Los datos de los programas**

# Los datos de los programas

## *Variabilidad de los datos*



# Identificadores

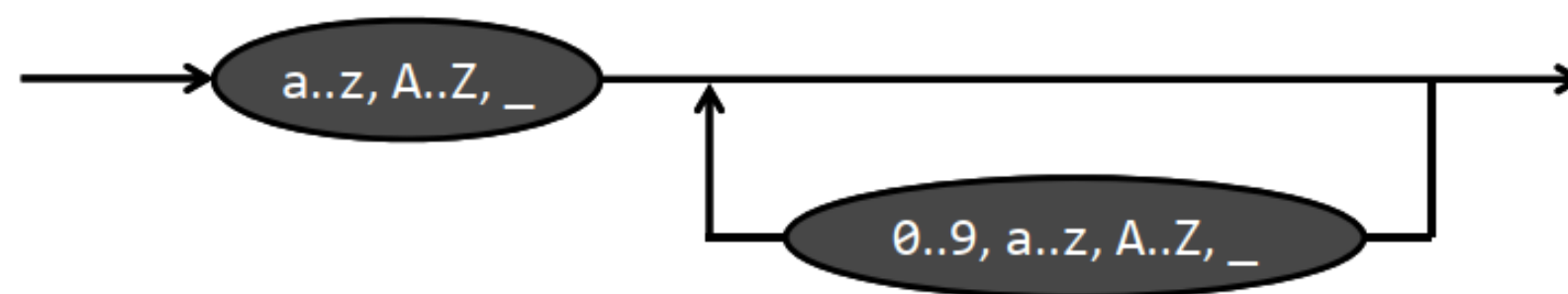
# Identificadores

≠ palabras reservadas

Para variables y constantes con nombre

- *Nombre* de un dato (para accederlo/modificarlo)
- Deben ser descriptivos

Sintaxis:

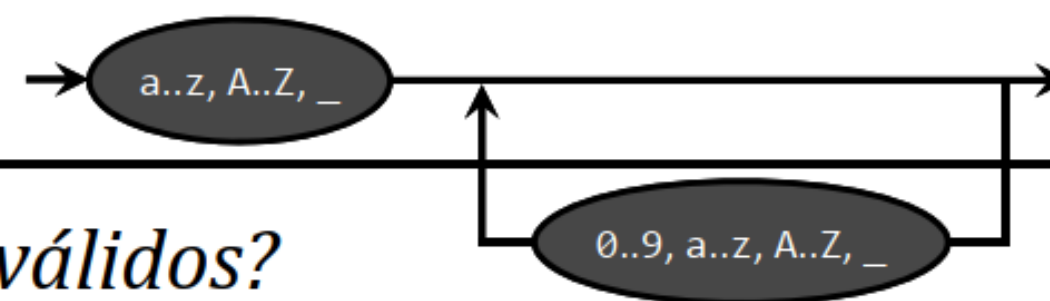


cantidad   prrecio   total   base   altura   area   numerador

Al menos 32 caracteres significativos

👁👁 ¡Ni eñes ni vocales acentuadas!

# Identificadores



*¿Qué identificadores son válidos?*

balance ✓

interesAnual ✓

\_base\_imponible ✓

años ✗

EDAD12 ✓

salario\_1\_mes ✓

\_\_edad ✓

cálculoNómina ✗

valor%100 ✗

AlgunValor ✓

100caracteres ✗

valor? ✗

\_12\_meses ✓

\_\_\_\_valor ✓

# **Tipos de datos**

# Tipos de datos

## Tipos

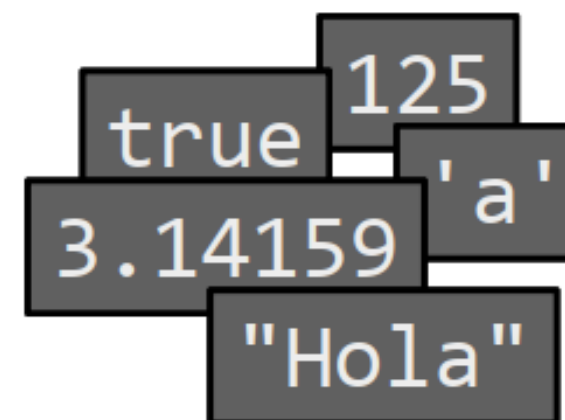
Cada dato, de un tipo concreto

Cada tipo establece:

- El conjunto (intervalo) de valores válidos
- El conjunto de operaciones que se pueden realizar

Expresiones con datos de distintos tipos (compatibles):

Transformación automática de tipos (*promoción de tipo*)



Anexo del Tema 2: detalles técnicos



## Tipos de datos básicos

---

int		
Números enteros (sin decimales)	1363, -12, 49	✓
float		
Números reales	12.45, -3.1932, 1.16E+02	
double		
Números reales (mayores intervalo y precisión)		✓
char		
Caracteres	'a' , '{', '\t'	
bool		
Valores lógicos (verdadero/falso)	true, false	
string		
Cadenas de caracteres (biblioteca string)	"Hola Mundo!"	
void		
<i>Nada</i> , ausencia de tipo, ausencia de dato ( <i>funciones</i> )		

# char

*Caracteres*

Intervalo de valores: Juego de caracteres (ASCII)

1 byte

Literales:

'a'    '%'    '\t'

Constantes de barra invertida (o *secuencias de escape*):

Caracteres de control

'\t' = tabulador    '\n' = salto de línea    ...



ASCII (códigos 32..127)



ISO-8859-1

(ASCII extendido: códigos 128..255)

# **bool**

## *Valores lógicos*

---

Sólo dos valores posibles:

- Verdadero (*true*)
- Falso (*false*)

Literales:

`true`    `false`

Cualquier número distinto de 0 es equivalente a `true`

El 0 es equivalente a `false`

## Mayúsculas y minúsculas

---

*C++ distingue entre mayúsculas y minúsculas*

`int`: palabra reservada de C++ para declarar datos enteros

`Int`, `INT` o `inT` no son palabras reservadas de C++

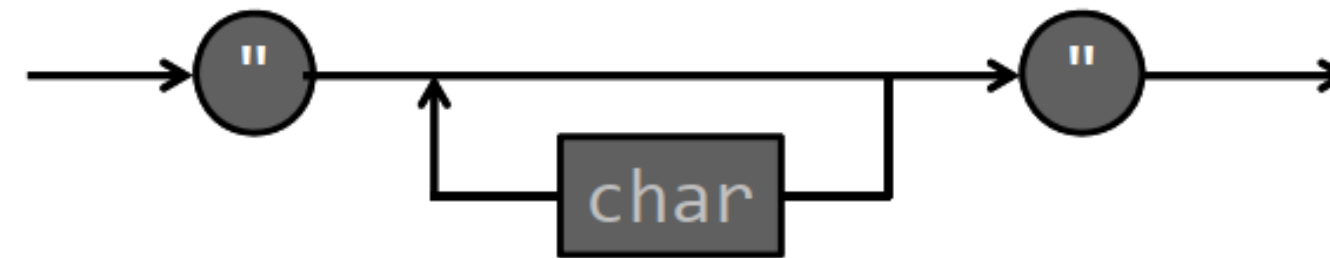
`true`: palabra reservada de C++ para el valor *verdadero*

`True` o `TRUE` no son palabras reservadas de C++

# string

## *Cadenas de caracteres*

"Hola" "Introduce el numerador: " "X142FG5TX?%A"



Secuencias de caracteres

Programas con variables de tipo string:

```
#include <string>
using namespace std;
```



Las comillas tipográficas (apertura/cierre) “...” NO sirven  
Asegúrate de utilizar comillas rectas: "..."

## Tipos de datos básicos: ejemplo

tipos.cpp

```
#include <iostream>
#include <string>
using namespace std; // Un solo using... para ambas bibliotecas

int main()
{
    int entero = 3; // Podemos asignar (inicializar) al declarar
    double real = 2.153;
    char character = 'a';
    bool cierto = true;
    string cadena = "Hola";
    cout << "Entero: " << entero << endl;
    cout << "Real: " << real << endl;
    cout << "Carácter: " << character << endl;
    cout << "Booleano: " << cierto << endl;
    cout << "Cadena: " << cadena << endl;

    return 0; }

¿Cuántos números hay en total en el programa?
¿Y caracteres? ¿Y cadenas? ¿Y booleanos?
```

```
D:\FP\Tema2>tipos
Entero: 3
Real: 2.153
Caracter: a
Booleano: 1
Cadena: Hola
D:\FP\Tema2>
```

```
D:/fb/16w95>
```



## Modificadores de tipos

---

- `signed` / `unsigned` : con signo (por defecto) / sin signo
- `short` / `long` : menor / mayor intervalo de valores

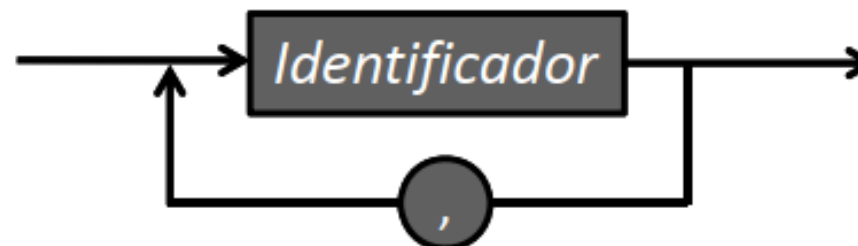
Tipo	Intervalo
<code>int</code>	<code>-2147483648 .. 2147483647</code>
<code>unsigned int</code>	<code>0 .. 4294967295</code>
<code>short int</code>	<code>-32768 .. 32768</code>
<code>unsigned short int</code>	<code>0 .. 65535</code>
<code>long int</code>	<code>-2147483648 .. 2147483647</code>
<code>unsigned long int</code>	<code>0 .. 4294967295</code>
<code>double</code>	<code>+ - 2.23e-308 .. 1.79e+308</code>
<code>long double</code>	<code>+ - 3.37E-4932 .. 1.18E+4932</code>

# **Declaración y uso de variables**



## Declaración de variables

[modificadores] tipo lista\_de\_variables;  
 └── Opcional ─┘

lista\_de\_variables → 

int i, j, l;

short int unidades;

unsigned short int monedas;

double balance, beneficio, perdida;



*Programación con buen estilo:*

Identificadores descriptivos

Espacio tras cada coma

Nombres de las variables en minúsculas

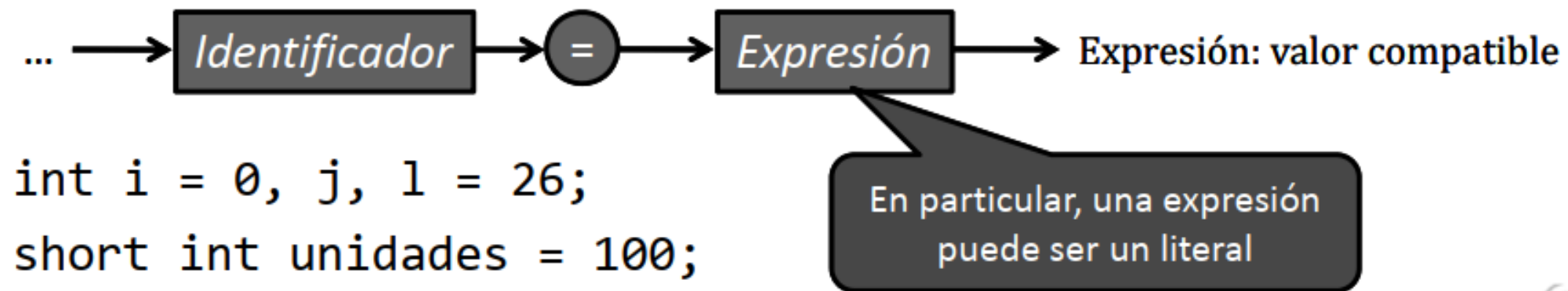
(Varias palabras: capitaliza cada inicial: interesPorMes)

# Inicialización de variables

*¡En C++ las variables no se inicializan automáticamente!*

*¡Una variable debe haber sido inicializada antes de ser accedida!*

Inicialización en la propia declaración:



## Uso de las variables

---

### *Obtención del valor de una variable*

- ✓ Nombre de la variable en una expresión

```
cout << balance;
```

```
cout << interesPorMes * meses / 100;
```

### *Modificación del valor de una variable*

- ✓ Nombre de la variable a la izquierda del =

```
balance = 1214;
```

```
porcentaje = valor / 30;
```

Las variables han de haber sido previamente declaradas

## Instrucciones de asignación

---

*El operador =*



A la izquierda, SIEMPRE una variable

```
int i, j = 2;
```

```
i = 23 + j * 5; // i toma el valor 33
```

# Instrucciones de asignación

---

## *Errores*

```
int a, b, c;
```

```
5 = a;
```

// ERROR: un literal no puede recibir un valor

```
a + 23 = 5;
```

// ERROR: no puede haber una expresión a la izda.

```
b = "abc";
```

// ERROR: un entero no puede guardar una cadena

```
c = 23 5;
```

// ERROR: expresión no válida (falta operador)

# Operadores

# Operadores

---

## *Operaciones sobre valores de los tipos*

Cada tipo determina las operaciones posibles

Tipos de datos numéricos (`int`, `float` y `double`):

- Asignación (=)
- Operadores aritméticos
- Operadores relacionales (menor, mayor, igual, ...)

Tipo de datos `bool`:

- Asignación (=)
- Operadores lógicos (Y, O, NO)

Tipos de datos `char` y `string`:

- Asignación (=)
- Operadores relacionales (menor, mayor, igual, ...)

# Operadores aritméticos

*Operadores para tipos de datos numéricos*

Operador	int	float / double
-	Cambio de signo	
+	Suma	
-	Resta	
*	Producto	
/	Div. entera	División real
%	Módulo	No aplicable
++	Incremento	
--	Decremento	



## Operadores aritméticos

---

*¿División entera o división real?*



Ambos operandos enteros: división entera

```
int i = 23, j = 2;
```

```
cout << i / j; // Muestra 11
```

Algún operando real: división real

```
int i = 23;
```

```
double j = 2;
```

```
cout << i / j; // Muestra 11.5
```

## Operadores aritméticos

*Módulo (resto de la división entera)*

%

Ambos operandos han de ser enteros

```
int i = 123, j = 5;
```

```
cout << i % j; // Muestra 3
```

División entera:

No se obtienen decimales → Queda un resto

$$\begin{array}{r}
 123 \quad | \quad 5 \\
 \hline
 24 \\
 \hline
 3
 \end{array}$$

123 % 5 → 3

# Operadores aritméticos

## *Operadores de incremento y decremento*

++/--

Incremento/decremento de la variable numérica en una unidad

Prefijo: Antes de acceder

```
int i = 10, j;
i=i+1; j = ++i; // Incrementa antes de copiar
j=i; cout << i << " - " << j; // Muestra 11 - 11
```

Postfijo: Después de acceder

```
int i = 10, j;
j=i; j = i++; // Copia y después incrementa
i=i+1; cout << i << " - " << j; // Muestra 11 - 10
```

 No mezcles ++ y -- con otros operadores

# Operadores aritméticos: ejemplo

```
#include <iostream>
using namespace std;

int main() {
    int entero1 = 15, entero2 = 4;
    double real1 = 15.0, real2 = 4.0;
    cout << "Operaciones entre los números 15 y 4:" << endl;
    cout << "División entera (/): " << entero1 / entero2 << endl;
    cout << "Resto de la división (%): " << entero1 % entero2 << endl;
    cout << "División real (/): " << real1 / real2 << endl;
    cout << "Num = " << real1 << endl;
    real1 = -real1;
    cout << "Cambia de signo (-): " << real1 << endl;
    real1 = -real1;
    cout << "Vuelve a cambiar (-): " << real1 << endl;
    cout << "Se incrementa antes (++ prefijo): " << ++real1 << endl;
    cout << "Se muestra antes de incrementar (posfijo ++): "
        << real1++ << endl;
    cout << "Ya incrementado: " << real1 << endl;
    return 0;
}
```

operadores.cpp

# Más sobre expresiones

## Orden de evaluación

---

*¿En qué orden se evalúan los operadores?*

$$3 + 5 * 2 / 2 - 1$$

¿De izquierda a derecha?

¿De derecha a izquierda?

¿Unos antes que otros?

Precedencia de los operadores (prioridad):

Se evalúan antes los de mayor precedencia

¿Y si tienen igual prioridad?

Normalmente, de izquierda a derecha

Paréntesis: fuerzan a evaluar su subexpresión

# Precedencia de los operadores

Precedencia	Operadores
Mayor prioridad	++ -- (postfijos)
	++ -- (prefijos)
	- (cambio de signo)
	* / %
Menor prioridad	+ -

$3 + 5 * 2 / 2 - 1 \rightarrow 3 + 10 / 2 - 1 \rightarrow 3 + 5 - 1 \rightarrow 8 - 1 \rightarrow 7$

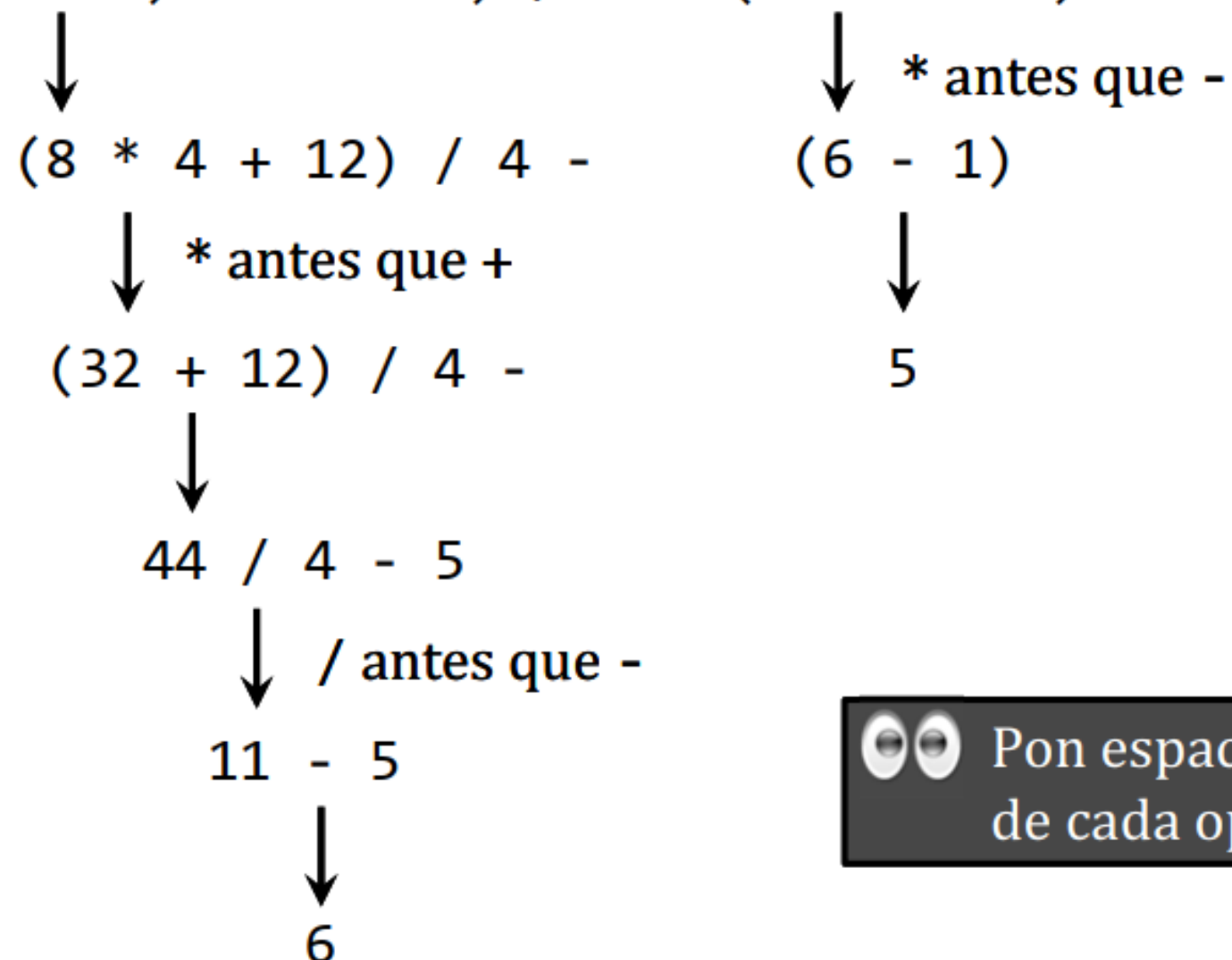
(Arrows under the first expression indicate left-to-right evaluation for  $*$  and  $/$ )  
 (An arrow under the second expression points to  $/$ )  
 (Arrows under the third expression indicate left-to-right evaluation for  $+$  and  $-$ )


Misma precedencia: Izquierda antes      Mayor precedencia      Misma precedencia: Izquierda antes



# Evaluación de expresiones

$((3 + 5) * 4 + 12) / 4 - (3 * 2 - 1)$  Primero, los paréntesis...



 Pon espacio antes y después de cada operador binario



## Una fórmula

fórmula.cpp

```
#include <iostream>
using namespace std;

int main()
{
    double x, f;
    cout << "Introduce el valor de X: ";
    cin >> x;
    f = 3 * x * x / 5 + 6 * x / 7 - 3;
    cout << "f(x) = " << f << endl;
    return 0;
}
```

$$f(x) = \frac{3x^2}{5} + \frac{6x}{7} - 3$$



Usa paréntesis para mejorar la legibilidad:

```
f = (3 * x * x / 5) + (6 * x / 7) - 3;
```

# **Operadores relacionales**

## **(condiciones simples)**

# Expresiones lógicas (*booleanas*)

## *Operadores relacionales*

Comparaciones (*condiciones*)

Condición simple ::= Expresión Operador\_relacional Expresión

Concordancia de tipo entre las expresiones

Resultado: bool (true o false)

<	menor que
<=	menor o igual que
>	mayor que
>=	mayor o igual que
==	igual que
!=	distinto de

### Operadores (prioridad)

...

\* / %

+ -

< <= > >=

== !=

= += -= \*= /= %=

# Operadores relacionales

---

Menor prioridad que los operadores aditivos y multiplicativos

```
bool resultado;
```

```
int a = 2, b = 3, c = 4;
```

```
resultado = a < 5;           // 2 < 5 → true
```

```
resultado = a * b + c >= 12; // 10 >= 12 → false
```

```
resultado = a * (b + c) >= 12; // 14 >= 12 → true
```

```
resultado = a != b;          // 2 != 3 → true
```

```
resultado = a * b > c + 5;    // 6 > 9 → false
```

```
resultado = a + b == c + 1;   // 5 == 5 → true
```



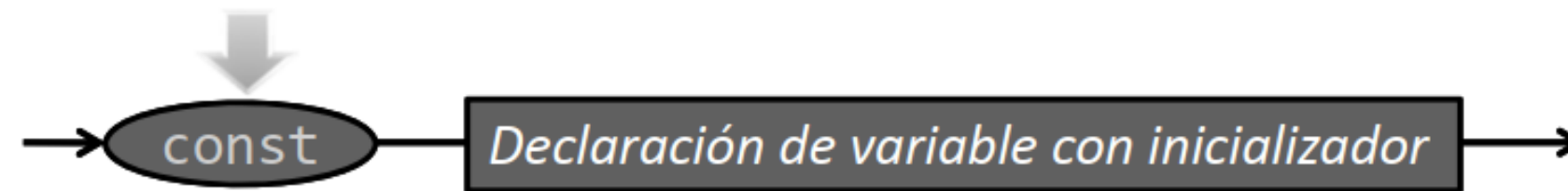
No confundas el operador de igualdad (==)  
con el operador de asignación (=)

# Constantes

# Constantes

*Declaración de constantes*    Modificador de acceso const

*Variables inicializadas a las que no dejamos variar*



```
const short int Meses = 12;
const double Pi = 3.141592,
    RATIO = 2.179 * Pi;
```

La constante no podrá volver a aparecer a la izquierda de un =



*Programación con buen estilo:*  
Pon en mayúscula la primera letra de una constante o todo su nombre

## ¿Por qué utilizar constantes con nombre?

- ✓ Aumentan la legibilidad del código

```
cambioPoblacion = (0.1758 - 0.1257) * poblacion;      VS.
cambioPoblacion = (RatioNacimientos - RatioMuertes) * poblacion;
```

- ✓ Facilitan la modificación del código

```
double compra1 = bruto1 * 18 / 100;
double compra2 = bruto2 * 18 / 100;
double total = compra1 + compra2;
cout << total << " (IVA: " << 18 << "%)" << endl;
```

3 cambios ←

```
const int IVA = 18;
double compra1 = bruto1 * IVA / 100;
double compra2 = bruto2 * IVA / 100;
double total = compra1 + compra2;
cout << total << " (IVA: " << IVA << "%)" << endl;
```

¿Cambio del IVA al 21%?  
1 cambio ←



# Constantes: ejemplo

constantes.cpp

```
#include <iostream>
using namespace std;

int main() {
    const double Pi = 3.141592;
    double radio = 12.2, circunferencia;
    circunferencia = 2 * Pi * radio;
    cout << "Circunferencia de un círculo de radio "
         << radio << ": " << circunferencia << endl;
    const double Euler = 2.718281828459; // Número e
    cout << "Número e al cuadrado: " << Euler * Euler << endl;
    const int IVA = 21;
    int cantidad = 12;
    double precio = 39.95, neto, porIVA, total;
    neto = cantidad * precio;
    porIVA = neto * IVA / 100;
    total = neto + porIVA;
    cout << "Total compra: " << total << endl;
    return 0;
}
```



# La biblioteca `cmath`

## Funciones matemáticas

```
#include <cmath>
```

Algunas ...	<code>abs(x)</code>	Valor absoluto de x
	<code>pow(x, y)</code>	x elevado a y
	<code>sqrt(x)</code>	Raíz cuadrada de x
	<code>ceil(x)</code>	Menor entero que es mayor o igual que x
	<code>floor(x)</code>	Mayor entero que es menor o igual que x
	<code>exp(x)</code>	$e^x$
	<code>log(x)</code>	Ln x (logaritmo natural de x)
	<code>log10(x)</code>	Logaritmo en base 10 de x
	<code>sin(x)</code>	Seno de x
	<code>cos(x)</code>	Coseno de x
	<code>tan(x)</code>	Tangente de x
	<code>round(x)</code>	Redondeo al entero más próximo
	<code>trunc(x)</code>	Pérdida de la parte decimal (entero)

# La biblioteca cmath

mates.cpp

```
#include <iostream>
using namespace std;
#include <cmath> ←
```

$$f(x, y) = 2x^5 + \frac{\sqrt{\frac{x^3}{y^2}}}{|x \times y|} - \cos(y)$$

```
int main() {
    double x, y, f;
    cout << "Valor de X: ";
    cin >> x;
    cout << "Valor de Y: ";
    cin >> y;
    f = 2 * pow(x, 5) + sqrt(pow(x, 3) / pow(y, 2))
        / abs(x * y) - cos(y);
    cout << "f(x, y) = " << f << endl;
    return 0;
}
```

pow() con argumento entero:

Usa el molde double():

pow(double(i), 5)



Pon un espacio detrás de cada coma en las listas de argumentos

# Operaciones con caracteres

char

Asignación, ++/ - - y operadores relacionales

*Funciones para caracteres (biblioteca ctype)*

isalnum(c)      true si c es una letra o un dígito

isalpha(c)      true si c es una letra

isdigit(c)      true si c es un dígito

islower(c)      true si c es una letra minúscula

isupper(c)      true si c es una letra mayúscula

false en caso contrario

toupper(c)      devuelve la mayúscula de c

tolower(c)      devuelve la minúscula de c



# Operaciones con caracteres

caracteres.cpp

```
...
#include <cctype>

int main() {
    char character1 = 'A', character2 = '1', character3 = '&';
    cout << "Carácter 1 (" << character1 << ").-" << endl;
    cout << "Alfanumérico? " << isalnum(character1) << endl;
    cout << "Alfabético? " << isalpha(character1) << endl;
    cout << "Dígito? " << isdigit(character1) << endl;
    cout << "Mayúscula? " << isupper(character1) << endl;
    character1 = tolower(character1);
    cout << "En minúscula: " << character1 << endl;
    cout << "Carácter 2 (" << character2 << ").-" << endl;
    cout << "Alfabético? " << isalpha(character2) << endl;
    cout << "Dígito? " << isdigit(character2) << endl;
    cout << "Carácter 3 (" << character3 << ").-" << endl;
    cout << "Alfanumérico? " << isalnum(character3) << endl;
    cout << "Alfabético? " << isalpha(character3) << endl;
    cout << "Dígito? " << isdigit(character3) << endl;
    return 0;
}
```

1 ≡ true / 0 ≡ false